UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Adaptación de aplicación para smartwatches para la regulación emocional de personas con TEA

Andrea Ruiz Pastor

Campus Internacional excelencia UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Adaptación de aplicación para smartwatches para la regulación emocional de personas con TEA

Autor: Andrea Ruiz Pastor
Tutor: Germán Montoro Manrique

Julio 2020

Andrea Ruiz Pastor
Adaptación de aplicación para smartwatches para la regulación emocional de personas con TEA

Andrea Ruiz Pastor

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Resumen

Taimun es una aplicación Android, distribuida entre Mobile y Wear, especialmente diseñada para personas con Trastorno del Espectro Autista (TEA), que facilita la autogestión emocional mediante el uso de estrategias regulatorias cuando el pulso cardíaco del portador de Wear sea elevado. Una regulación emocional consiste de una o más estrategias cuyo objetivo es restaurar el estado anímico normal, como por ejemplo respirar profundamente.

Previamente a nuestro trabajo, Taimun era incompatible con las versiones actuales de Android. Nuestra primera tarea consistió en la adaptación a SDKs actuales de Android. Las mayores incompatibilidades consistían en el uso de librerías AppCompat deprecadas y el uso de patrones de diseño no permitidos, como BroadcastListener globales o la exposición de File URIs.

Una vez adaptada la aplicación, ampliamos su funcionalidad permitiendo que desde Mobile se pudiera especificar un umbral de pulsaciones cardíacas distinto para cada regulación emocional. De esta manera aumentamos el potencial de personalización. Adicionalmente añadimos la posibilidad de iniciar y parar regulaciones emocionales de manera remota. Previamente las regulaciones debían de iniciarse manualmente pulsando un botón «START» en el Wear, pero con esta mejora, esto se puede lograr a distancia mediante Mobile. Una última mejora añadida es el sharing de regulaciones emocionales entre aplicaciones móvil. Para ello, hemos usado Firebase y Google Cloud. Al compartir una regulación se presenta un código que luego puede ser introducido en otro dispositivo para la descarga de la regulación. Después de la adaptación e implementación, procedimos a la realización de pruebas, tanto de las diferentes funcionalidades por separado, como de la app en su conjunto.

Palabras clave

Aplicación, smartphone, smartwatch, TEA, autismo, regulación emocional, android, firebase, taimun

Abstract

Taimun is an Android application, distributed over Mobile and Wear versions, that is specifically designed for users with Autistic Spectrum Disorder (ASD). It eases emotional self-management with the use of regulatory strategies that are started when the cardiac pulse rate is high. An emotional regulation is composed of one or more strategies whose aim is to restore the normal emotional state. For example, deep breathing constitutes one such strategy.

Before our work, Taimun was incompatible with current versions of Android. Our first task implicated the adaptation to current Android SDKs. The greatest incompatibilities were due to the use of deprecated AppCompat libraries, as well as design patterns that were no longer allowed, like global BroadcastListeners or the exposure of File URIs.

Once the application was adapted, we extended its functionality by allowing the specification of a different custom cardiac pulse threshold for each emotional regulation, increasing the customizing potential. Additionally, we added the possibility of starting and stopping emotional regulations remotely. Previously, regulations had to be initiated manually by pressing a «START» button on Wear, but with this improvement this can be achieved remotely via Mobile. A further improvement added was the sharing of emotional regulations between Mobile Apps. This was implemented using Google Cloud as well as Firebase. When a regulation is shared a code is presented that serves as input in another device that will subsequently download the regulation. Finally, we proceeded to testing both the individual functionalities added as well as the application as a whole.

Keywords

Application, smartphone, smartwatch, ASD, autism, emotional regulation, android, firebase, taimun

ÍNDICE

Listas

Lista de figuras

1

Introducción

El objetivo de este trabajo de fin de grado es adaptar y ampliar la funcionalidad de una aplicación ya existente, Taimun, para que pueda volver a ser utilizada mediante su descarga en Play Store.

Taimun es una aplicación que sirve para la autorregulación emocional de personas con TEA basada en smartwatches, que toma las pulsaciones de los usuarios y al sobrepasarse un umbral muestra una serie de regulaciones en forma de imágenes, pictogramas y demás con el objetivo de que el portador logre retomar un estado de calma. La aplicación wear es sincronizada y controlada por la aplicación móvil, de manera que el usuario de la segunda puede controlar el estado del portador del reloj, consultando sus estadísticas y seleccionando las regulaciones más adecuadas a su estado en cada momento.

1.1. Trastorno del espectro autista

El trastorno del Trastorno del Espectro Autista (TEA) es una afección neurológica multifacética sin causa genética determinada, que afecta al intelecto, comunicación y comportamiento. Aunque puede ser diagnosticado a cualquier edad, se considera un trastorno de desarrollo ya que, generalmente, los síntomas aparecen en los dos primeros años de vida. Algunos síntomas incluyen:

- Dificultad para comunicarse e interactuar con otras personas
- Intereses reducidos y comportamientos repetitivos
- Síntomas que dificultan la habilidad para funcionar correctamente en la escuela, trabajo u otras áreas de la vida

El TEA impone una gran carga tanto sobre los propios pacientes como a sus padres o guardianes. Por ejemplo, la esperanza de vida de las personas con TEA es significativamente menor que la de sus equivalentes neurotípicos. Lo mismo ocurre con la inserción laboral y otros índices de calidad de vida como depresión. Por otra parte, muchos padres se preguntan sobre el futuro de sus hijos con TEA una vez que ellos sean incapaces de prestar cuidados. Los síntomas concretos del TEA varían mucho de persona a persona, lo cual dificulta la investigación patofisiológica, y manda a la división

en subgrupos a la hora de realizar estudios bioquímicos, retrasando y encareciendo la adquisición de nuevo conocimiento acerca del TEA.

Actualmente no existe ninguna terapia curativa, y el tratamiento se limita a aliviar síntomas concretos con medicaciones que muchas veces tienen efectos secundarios negativos. Intervenciones psíquico-conductales dedicadas a mejoras emocionales y afectivas no suelen realizarse de manera universal, y además tienen un alto coste económico para las familias de personas con TEA.

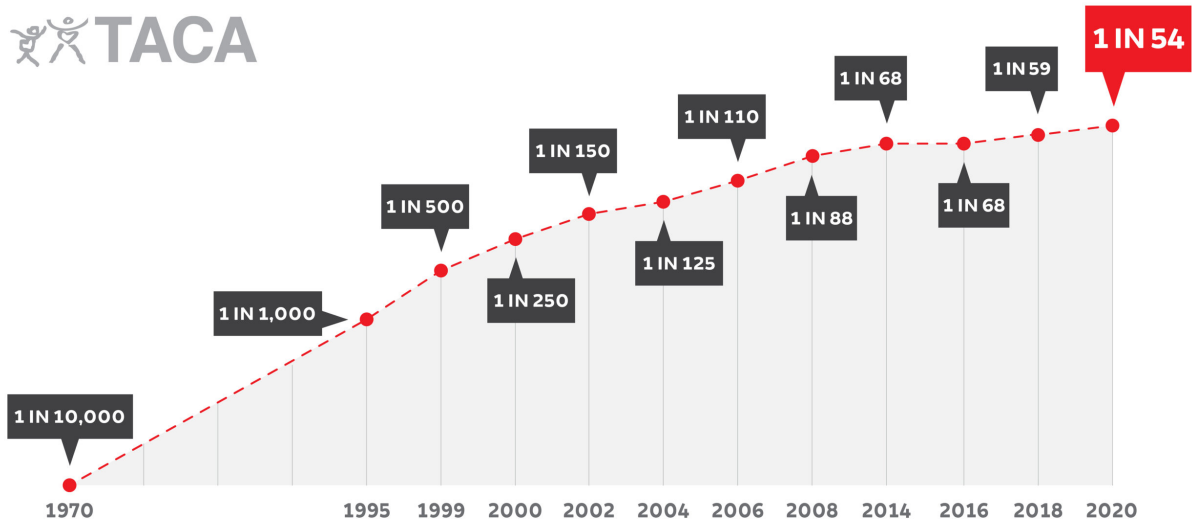


Figura 1.1: Prevalencia del TEA en Estados Unidos

1.2. Taimun

Taimun responde a la necesidad de autorregulación emocional por parte de personas con TEA. Podemos identificar dos componentes principales de Taimun: la aplicación móvil y la aplicación watch. La primera está destinada a ser usada por el guardián o persona de apoyo, mientras que la segunda es instalada en un smartwatch Android portado por la persona con TEA.

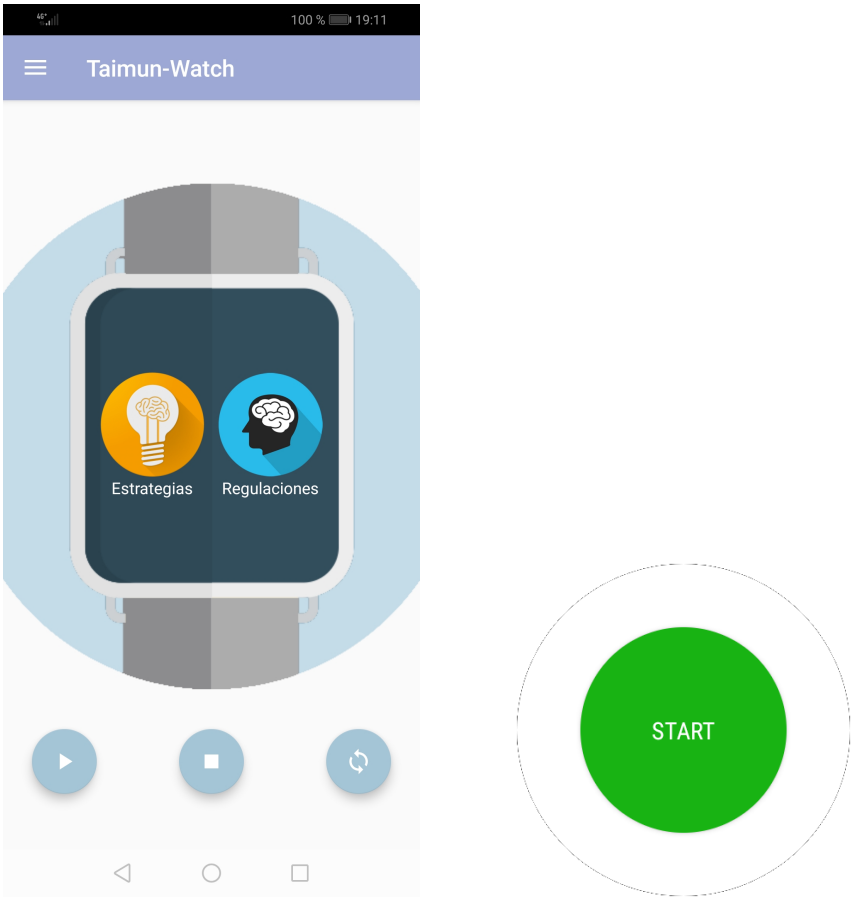
El guardián, desde la app móvil, puede enviar regulaciones emocionales de manera individualizada a los distintos relojes. Una vez que las regulaciones estén cargadas en los relojes, la subida de pulso cardíaco en el portador del watch dispara la regulación. El objetivo es que la regulación restaure el estado emocional normal del portador.

El reloj se puede llevar bloqueado, y cuando se deba iniciar una regulación se precede la ejecución de la misma por una vibración que alerte al portador. La regulación aparece entonces en pantalla utilizando sencillos pictogramas (o texto) que muestran la acción a realizar. Cada pictograma puede requerir que el usuario indique la realización de su tarea asociada (pulsando sobre el reloj), o bien aparecer

durante un umbral de tiempo predeterminado, que puede ir indicado por un overlay semi-transparente que desciende uniformemente por la pantalla, o por una circunferencia que va desapareciendo, aportando una dimensión temporal.

Una regulación emocional está compuesta por una serie de estrategias, que normalmente están en correspondencia directa con los pictogramas mostrados, y suelen referirse a sencillas acciones a realizar. Cada estrategia aborda el problema de restaurar la calma emocional desde un punto de vista distinto, y es aquí donde la experiencia del personal de apoyo permite individualizar la regulación emocional mediante la apropiada elección de estrategias.

A modo de ejemplo podemos crear una regulación nombrada como «Respira profundamente» que emplee como estrategias la inspiración y expiración profunda. Este tipo de respiración suele iniciar un movimiento diafragmático que activa el sistema nervioso parasimpático, reduciendo la agitación emocional. Por otra parte, podemos crear otra que se llame «Salta y Corre», que esté formada por estrategias de movimiento activo, de alta intensidad y corta duración que permita la liberación de endorfinas en el Sistema Nervioso Central (SNC) .



(a) Taimun Mobile (b) Taimun Watch

Figura 1.2: Home Taimun

1.3. Motivación

Actualmente existe una gran necesidad de encontrar tratamientos efectivos frente al TEA. Debido al largo ciclo de desarrollo de productos farmacológicos, no es realista esperar a corto plazo una solución en este frente. En cambio, sí podemos abordar ciertas características del TEA como la regulación emocional, logrando cierto alivio sintomático. Es en esta categoría en la que se encuentra Taimun. Como informáticos, es nuestro deber usar los conocimientos adquiridos a lo largo de nuestra formación para mejorar la calidad de vida de las personas más desaventajadas y en riesgo de exclusión social, como es el caso de las personas con TEA.

En el estado en que nos encontramos la aplicación, ésta era inservible debido al uso de librerías que estaban deprecadas así como patrones de diseño no permitidos por las versiones de Android más modernas. Por otra parte, después de inspeccionar el código fuente, llegamos a la conclusión de que la mayoría de la funcionalidad programada funcionaría con adecuadas intervenciones de refactorización y modernización.

Además, a la hora de analizar la funcionalidad de Taimun se echaron en falta funcionalidades que permitieran un mayor control del umbral de activación de las regulaciones emocionales y la posibilidad de compartir regulaciones entre diferentes usuarios de la aplicación móvil, como había sido sugerido en versiones anteriores sin haberse llevado a cabo. Además, nos pareció de gran interés dotar a la aplicación móvil de la capacidad de comenzar o finalizar regulaciones de manera remota.

El objetivo final es poder publicar Taimun en Play Store para que vuelva a estar disponible y pueda ser utilizada con las últimas versiones de Android.

1.4. Objetivos

Este trabajo tiene dos objetivos principales. Por una parte, adaptar la aplicación a las versiones modernas de Android, para que pueda ser nuevamente utilizada y mejore la calidad de vida de personas con TEA. Para que esto sea posible, es un paso muy importante subirla a Play Store, para que cualquier persona pueda beneficiarse de ella. Adicionalmente, planteamos las siguientes mejoras por encima de la funcionalidad ya existente:

- Umbral de pulsaciones personalizado: el usuario que utiliza la aplicación móvil puede introducir un umbral de pulsaciones cardíacas personalizado para cada regulación. Con esta mejora se pretende ajustar aún más las regulaciones emocionales a las necesidades individuales de los portadores del reloj.
- Inicio y fin de asistencia remoto: el usuario que utiliza la aplicación móvil puede empezar y terminar una medición de pulsaciones en el reloj, de manera que cambiar de regulación sin

la intervención del usuario del reloj.

- Compartir regulaciones: Un usuario de Taimun Mobile puede compartir regulaciones con otro usuario. De esta manera si una regulación resulta especialmente útil se puede pasar a otros dispositivos sin necesidad de que se tenga que crear de cero.

1.5. Organización de la memoria

La memoria consta de los siguientes capítulos:

1.– Introducción: Aportamos una visión general de los motivos que inspiraron la realización del trabajo, listando los objetivos y dando una breve descripción de la aplicación.

2.– Tecnología a utilizar: En esta sección detallamos la tecnología utilizada para el desarrollo del proyecto, comenzando con una visión general de Android en sus facetas de Mobile y de Wear, el potencial que aporta conectar ambos dispositivos, y las tecnologías de desarrollo que hemos usado, entre las que destaca Android Studio, un IDE proporcionado por Google, y adb una herramienta de línea de comandos que permite el despliegue rápido de aplicaciones Wear.

3.– Diseño: En esta sección indicamos la arquitectura de las aplicaciones Mobile y Watch, mostramos los puntos que impedían la ejecución del código ya presente y planteamos cómo introducir las mejoras planteadas previamente

4.– Desarrollo: Detallamos los pasos seguidos para adaptar las aplicaciones a versiones actuales así como para incluir funcionalidad adicional, describimos las múltiples dificultades encontradas y las soluciones que encontramos.

5.– Integración, pruebas y resultados: Describimos las pruebas realizadas para garantizar que la aplicación fue adaptada con éxito a las versiones actuales de Android y que las nuevas mejoras introducidas se comportan de manera esperada. Las pruebas comenzaron siendo unitarias, comprobando el funcionamiento individual de los módulos adaptados o introducidos, y luego progresan a pruebas de integración, donde se comprueba la correcta conexión entre las aplicaciones Mobile y Wear. Las ultimas pruebas consisten en «Real World Testing» donde se prueba Taimun en circunstancias que aproximen su uso real.

6.– Conclusiones y trabajo futuro: Resumimos el trabajo realizado y planteamos posibles direcciones a seguir en el futuro para mejorar más la calidad de Taimun.

2

Tecnología a utilizar

2.1. Android

Taimun utiliza el sistema operativo Android. Las ventajas de usar Android incluyen el bajo coste de desarrollo con respecto a otros sistemas operativos de dispositivos móviles como iOS, así como una amplísima y exhaustiva documentación, que permite un desarrollo acelerado.

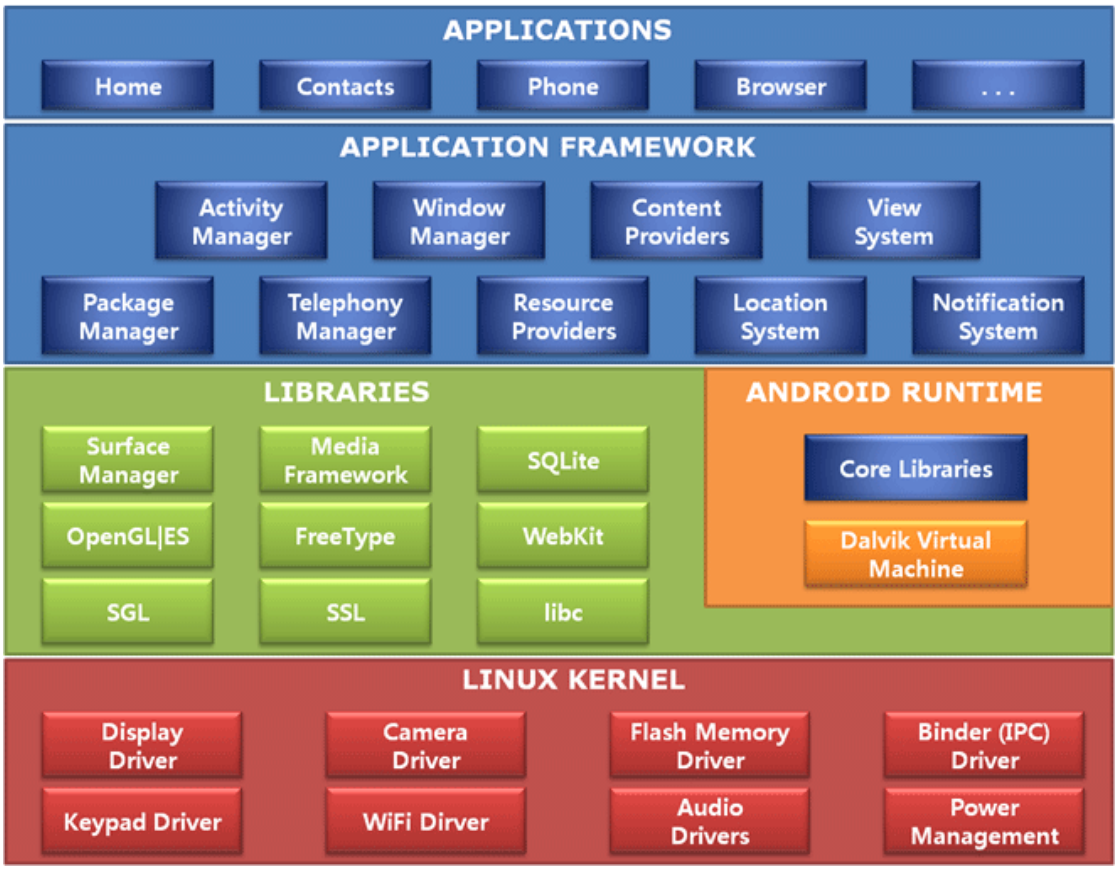


Figura 2.1: Componentes arquitectura Android

Dentro de Android, y de carácter relevante para Taimun, se encuentra la versión Mobile, que fue la primera desarrollada, y Wear, una versión de Android especialmente adaptada para wearables. Los

wearables son dispositivos electrónicos especialmente diseñados para ser portados de manera poco intrusiva por un usuario, y se suelen caracterizar por su capacidad de obtener datos biométricos.

Android ofrece una robusta API para conectar un dispositivo móvil (Mobile) con un wearable (Wear) mediante bluetooth. Taimun hace un fuerte uso de dicha API.

Normalmente, un wearable se empareja con un dispositivo móvil en una relación N-1. Es decir, varios wearables se pueden vincular con un dispositivo móvil, pero un wearable puede conectarse con un único móvil.

Una vez que el wearable esté conectado, puede recibir paquetes enrutados. Estos paquetes son de bajo peso, por lo que en ocasiones se debe de pasar más de un paquete para transmitir la totalidad de la información requerida. El wearable también puede enviar paquetes «respuesta» al dispositivo móvil, con lo que la API permite establecer una conexión bidireccional.

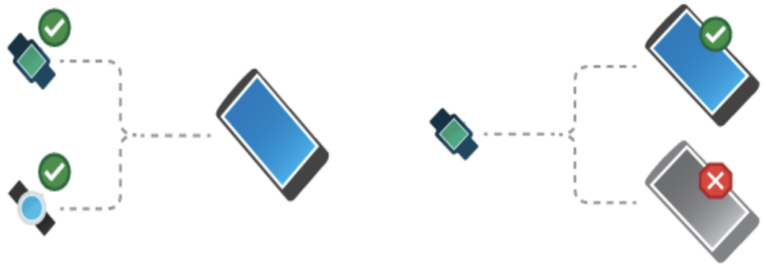


Figura 2.2: Configuraciones entre dispositivos wear y mobile

2.2. Dispositivos

Para poder probar desde el comienzo Taimun en un entorno que se asemejara al de producción, compramos un wearable android con capacidad para leer el pulso cardíaco de marca Vapor. Por otra parte hemos usado varios dispositivos móviles de diversos fabricantes: Huawei, Xiaomi y Samsung.

Figura 2.3: Dispositivos utilizados

2.3. Herramientas

Android ofrece herramientas de desarrollo que incluyen gran cantidad de funcionalidades para facilitar el desarrollo. Cabe destacar Android Studio, un IDE integrado que permite realizar compilaciones, despliegues, debugs, pruebas y gestiones de dispositivos conectados al ordenador.

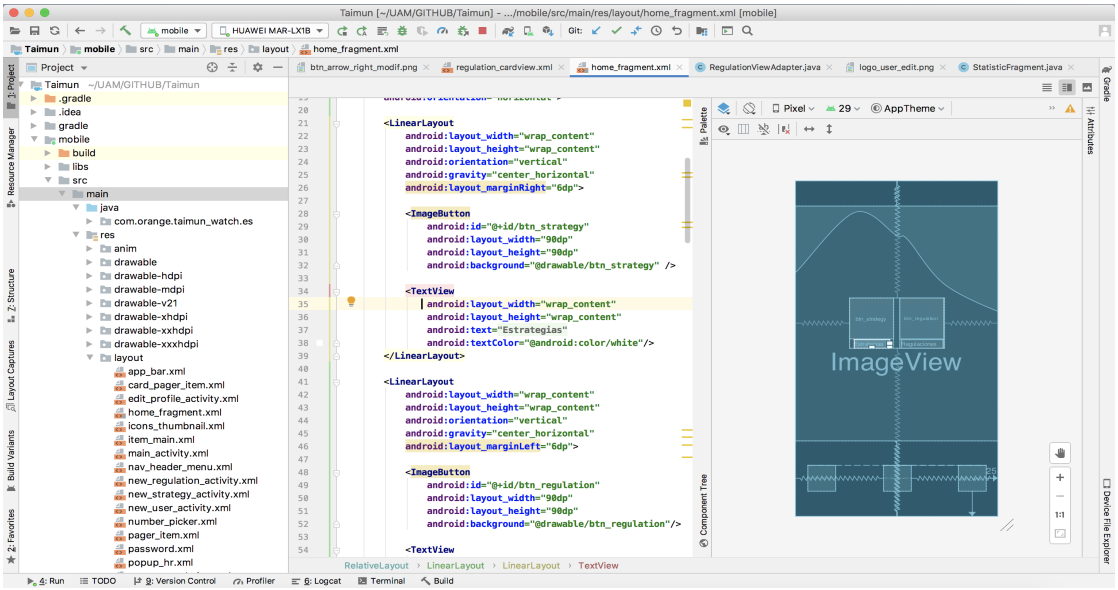


Figura 2.4: Android Studio

Hacemos énfasis en el debugger de Android, que depura incluso hilos asíncronos mediante su capacidad de «attach to process». Como veremos más adelante, este depurador fue crítico para entender los detalles del funcionamiento de Taimun.

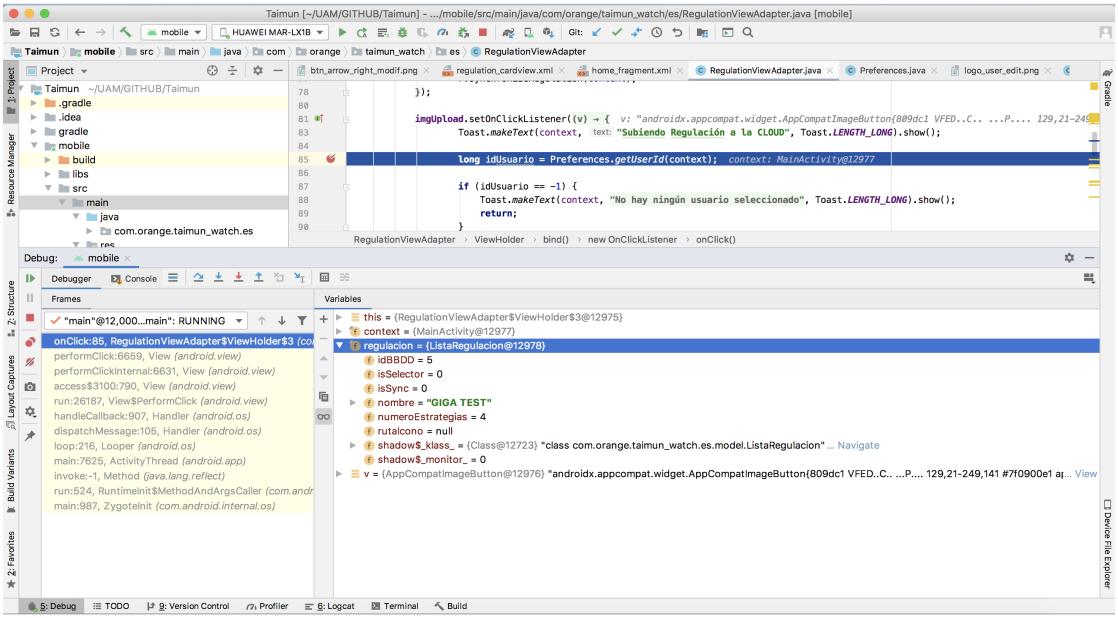


Figura 2.5: Android Studio Debugger

Adicionalmente hemos hecho uso de adb (android debug bridge). Este ejecutable de línea de comando es necesario para conectar el wearable al ordenador. Esto se debe a que el reloj wearable que compramos no tenía puerto mini-USB, con lo que no se podría conectar de manera directa al ordenador. Para solventar este problema adb permite establecer un «forwarding» de puertos, de manera que un puerto bluetooth del móvil (conectado mediante mini-USB al ordenador) es conectado al reloj, y de esta manera se establece una conexión entre el ordenador (y por lo tanto Android Studio) y el reloj.

Por otra parte, adb ayuda a realizar despliegues cuando la funcionalidad estándar de despliegue de Android Studio falla. Normalmente los despliegues pueden realizarse directamente desde Android Studio, pero debido al alto peso de la App Wear, así como la baja capacidad del reloj que adquirimos, adb fue usado para acortar los tiempos de despliegue, que en caso contrario llegaban a alcanzar los 15 minutos.

2.4. Google Cloud

Para poder permitir que los usuarios compartan datos de regulaciones necesariamente tiene que haber un componente de tecnología Web. Esto es porque las regulaciones, al estar formadas por varios archivos de distinto tipo no se pueden compartir fácilmente por medios tradicionales como email. Nos planteamos usar un servidor propio, pero esto tiene dos desventajas principales: por una parte, los servidores propios requieren de mantenimiento continuo y actualizaciones de los distintos paquetes que usan. Por otra parte, un servidor es difícil de escalar en caso de que haya más usuarios de lo previsto inicialmente.

Debido a estos problemas, optamos por alojar el componente de compartir datos en la Cloud. De los tres grandes proveedores existentes: AWS, Google Cloud e IBM, optamos por Google Cloud ya que es el que ofrece una mayor facilidad de uso y mejores precios.

Google Cloud consta de una serie de servicios distribuidos y mantenidos por Google, lo cual facilita mucho su uso, y los presenta en una capa de abstracción que hace énfasis en la utilidad al programador en vez de al mantenimiento estéril de la tecnología subyacente.

De Google Cloud vamos a utilizar tres componentes: Firebase Realtime Database, Cloud Storage y Cloud Functions.

Firebase Realtime Database

Es un servicio de almacenamiento de datos en real time que permite el acceso rápido a datos primitivos. Originalmente estaba en una estructura de BBDD noSQL clásica, es decir, como un gran objeto JSON. En nuevas iteraciones ha sido sustituido por un modelo híbrido que permite realizar consultas y crear índices. Tiene la ventaja de que es muy escalable y permite crear reglas de seguridad

de manera expresiva.

Después de analizar las estructuras de datos presentes en Taimun, concluimos que la representación más directa sería en la Realtime Database con implementación JSON.

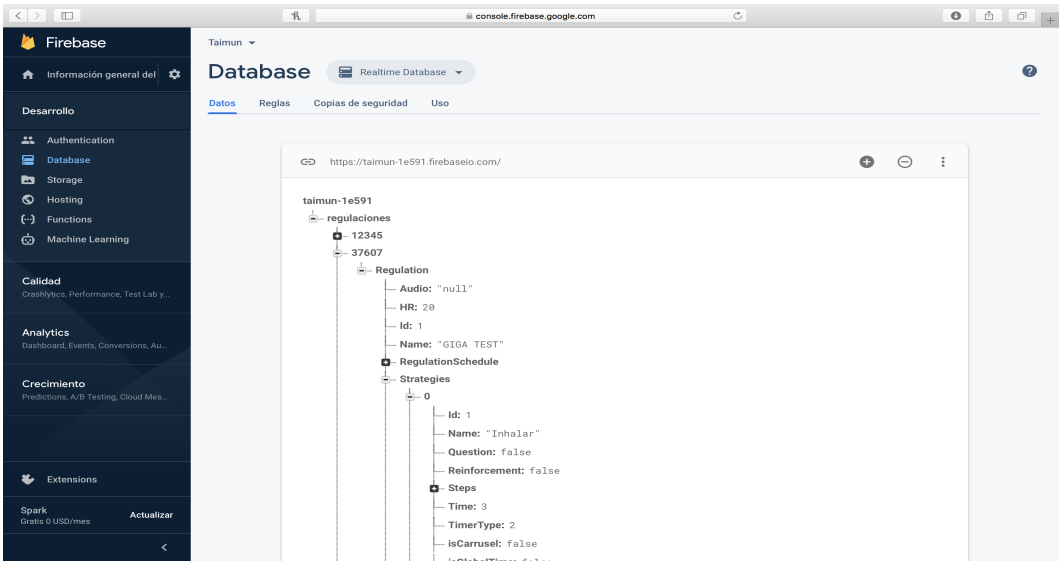


Figura 2.6: Realtime Database

Google Cloud Storage

Para almacenar datos de la regulaciones que no sean almacenables en Firebase Realtime Database usaremos Google Cloud Storage que permite el almacenamiento de datos arbitrarios. Estos datos se pueden referenciar mediante un sencillo sistema de enrutación, muy parecido al de cualquier sistema de archivos moderno, excepto que está en la nube.

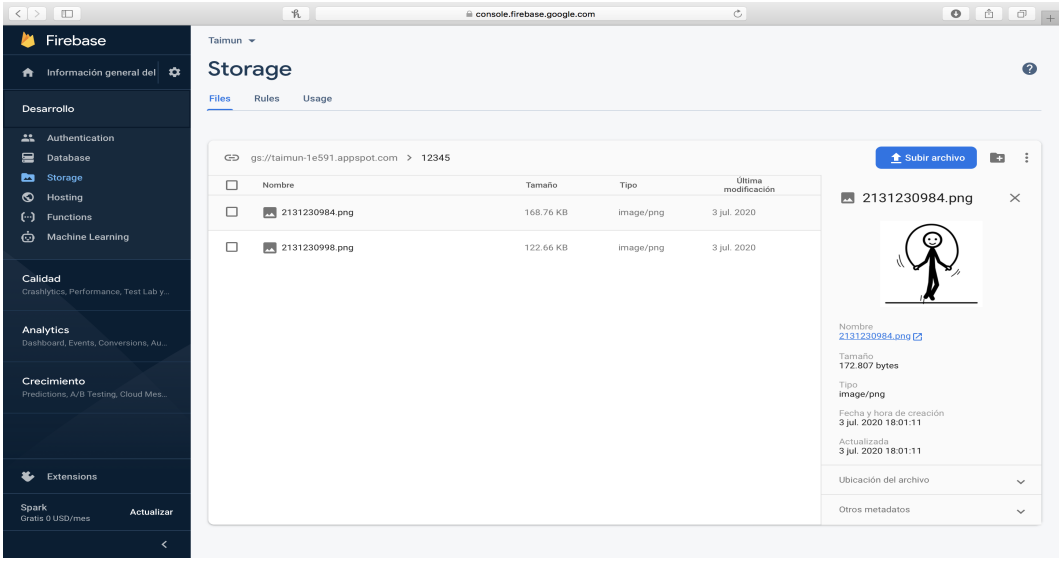


Figura 2.7: Google Cloud Storage

Google Cloud Functions

Usaremos funciones ejecutables en un entorno cloud cuando haya que ejecutar código independientemente del Mobile o Wear Taimun. Esto será útil para gestionar la base de datos. Las Cloud Functions, de entre las cuales AWS fue pionera, suponen un nuevo concepto a la hora de arquitecturizar aplicaciones que requieran realizar uso de comunicación mediante internet, ya que aparte de ser muy escalable, requiere mantenimiento mínimo.

Para que la base de datos no llegue a tener demasiada información y evitar que ésta quede desfasada, implementamos la función deleteAll, la cual está programa en Google Cloud Scheduler para ejecutarse todos los días a las 5am, y se muestra en la figura 2.8.

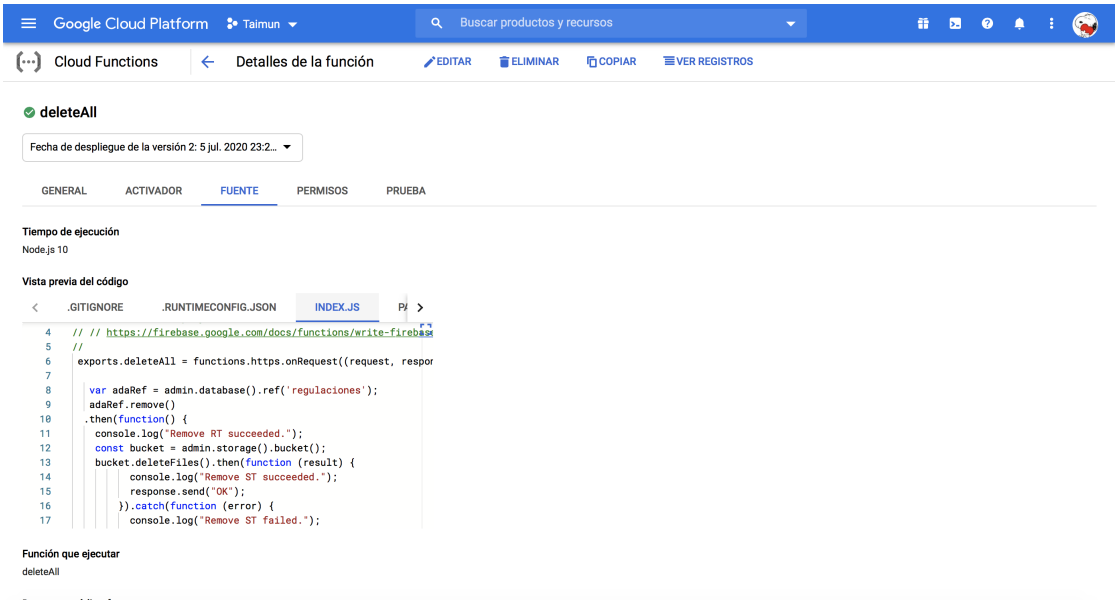


Figura 2.8: Google Cloud Function

2.5. Play Store

Para que la aplicación volviese a estar en Play Store, el código fue enviado a la Fundación Orange, ya que es quien se encargó de las primeras versiones, así que nuestra labor se limitó a asegurarnos de que se cumpliesen todos los requisitos necesarios para ello (tamaño máximo, firma de la aplicación y requisitos de APK y API). Esperamos que dentro de poco se encuentre subida y que pueda volver a ser utilizada.

3

Diseño

En esta sección nos centramos en el diseño de la funcionalidad adicional, así como la planificación del proceso de refactorización y actualización de Taimun.

3.1. Actualización de Taimun

Nuestro primer objetivo fue recuperar la totalidad de la funcionalidad de Taimun ya programado. Los pasos que planteamos fueron los siguientes:

- Inspección de código.
- Primer despliegue en un dispositivo.
- Ejecución comprobando qué funcionalidades operan correctamente y cuáles no.
- Ejecución en detalle de distintas funcionalidades mediante el depurador de Android: En esta etapa comenzamos el bosquejo de diagramas de flujo y de clase para profundizar nuestra comprensión de la app. Estos diagramas, debido a la naturaleza profundamente asíncrona de la app, fueron muy útiles a la hora encontrar distintos puntos de fallo.
- Identificación de puntos de fallo: En esta etapa también se realiza una clasificación de todos los errores encontrados. Esto es importante para priorizar los errores críticos, y también permite agrupar errores aparentemente dispares en clases que respondan a un único arreglo o a varios arreglos idénticos, reduciendo la complejidad.
- Experimentación con posibles soluciones, favoreciendo aquellas que tengan pocos efectos secundarios y aumenten lo mínimo posible la complejidad ciclomática.

3.2. Nuevas Funcionalidades

Como se ha descrito anteriormente, las tres mejoras principales que queríamos añadir son la capacidad de introducir un umbral personalizado para cada regulación, la capacidad de activar y desactivar

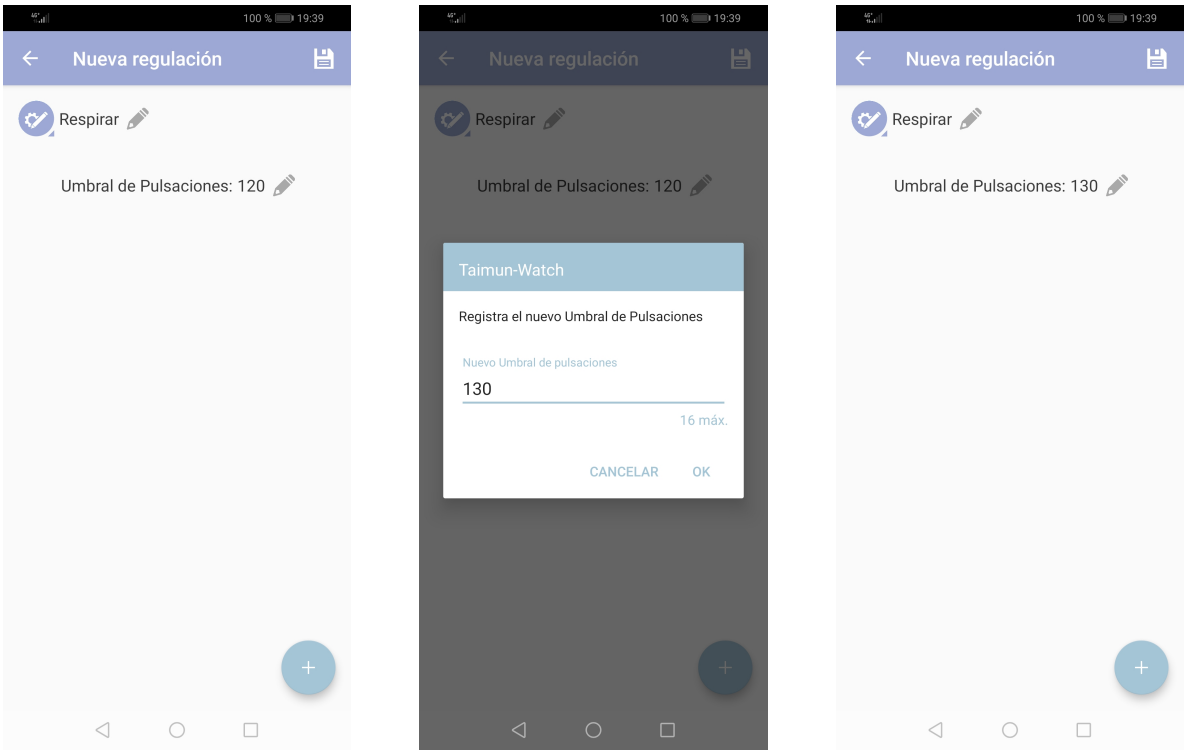
el proceso de activación de regulación emocional de manera remota y la capacidad de compartir regulaciones entre diferentes dispositivos móviles.

3.2.1. Umbral Personalizable

Inicialmente contemplamos la posibilidad de personalizar el umbral de activación de estrategia para cada usuario, pero después de reflexionar, llegamos a la conclusión que permitir un umbral distinto para cada regulación aumentaría mucho más las posibilidades de personalización.

El umbral se introduciría como un campo adicional en la Card de Regulación, y posteriormente sería guardado en un nuevo campo dentro de la base de datos miniSQL que almacena todos los datos relacionados con regulaciones.

Una vez que la regulación se envía por bluetooth en formato JSON al Wear, el umbral se transmitiría como un campo adicional. A la recepción y parseo del JSON, el Wear almacenaría el umbral como una preferencia que permita acceso rápido al proceso que determina si hay que iniciar la regulación emocional.



(a) Creación de nueva regulación (b) Edición umbral de pulsaciones (c) Nuevo umbral definido

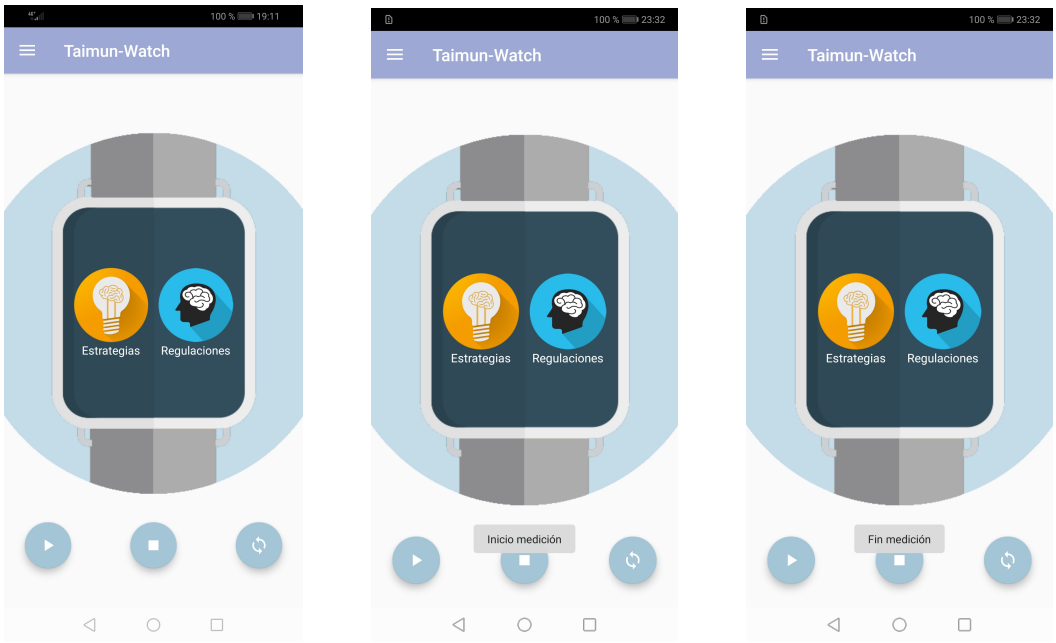
Figura 3.1: Interfaz umbral personalizable. Para definir el umbral para una regulación, el usuario debe hacer click en el icono de edición en la figura 3.1(a), con lo cual se le abriría el cuadro de edición de la figura 3.1(b), en el cual puede introducir el número deseado. Al seleccionar «OK» el parámetro se actualiza de manera correcta, como se ve en la figura 3.1(c)

3.2.2. Inicio y Parada Remota

Que el guardián de una o varias personas con TEA que estén portando Taimun Wear pueda comenzar y finalizar la regulación de manera remota añade una nueva dimensión de flexibilidad. Anteriormente a este trabajo, el reloj debía de iniciarse y pararse pulsando un botón. Esto puede suponer un problema si se está gestionando un gran grupo.

Por otra parte, una nueva regulación sólo puede cargarse cuando el reloj esté parado. Esto disminuye la capacidad de reacción de los guardianes, puesto que si perciben un cambio emocional que requiera una nueva regulación, han de parar físicamente el reloj. Con esta nueva mejora ambos escenarios se pueden realizar de manera remota desde Taimun Mobile.

Decidimos posicionar ambos botones de comienzo y finalización en la pantalla principal, y que tengan efecto sobre el Wear que actualmente esté seleccionado como activo. También concluimos que la funcionalidad de empezar y acabar fuera idempotente, es decir, que el pulsado en repetidas ocasiones lleve al mismo resultado que pulsar una única vez. Este patrón de diseño simplificó el desarrollo a la vez que minimizó los posibles efectos secundarios.



(a) Home (b) Inicio remoto (c) Parada remota

Figura 3.2: Inicio y parada remoto. Al hacer click sobre el botón de iniciar medición remota (representado con el símbolo «play» en la figura 3.2(a)), se inicia la medición en el reloj y se muestra el mensaje de la figura 3.2(b). Análogamente, al pulsar el botón de parar la medición (representado con el símbolo «stop»), se finaliza la medición y se muestra el mensaje de la figura 3.2(c)

Una vez que el reloj reciba el comando de inicio o de parada, ha de reaccionar apropiadamente. Para ello decidimos crear dos BroadcastReceiver; StartReceiver y StopReceiver, que ejecuten fun-

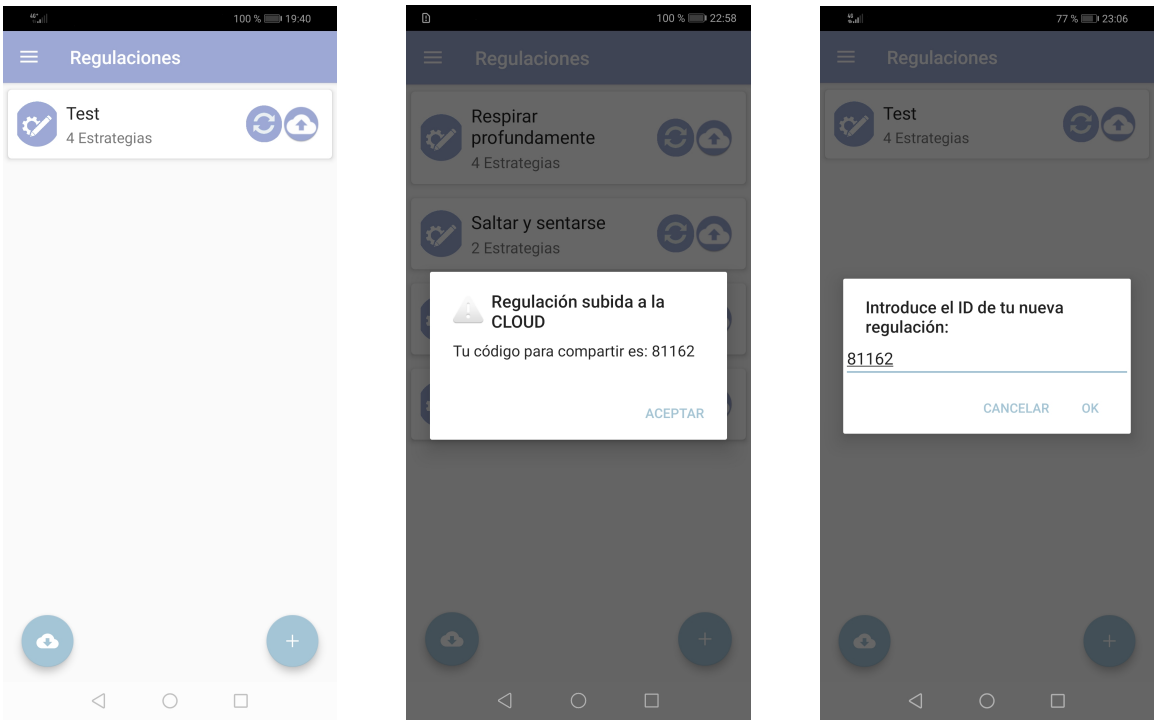
ciones que comiencen o finalicen la regulación. El desacoplamiento de la recepción de señal y su consecuencia simplifica adicionalmente el desarrollo.

3.2.3. Compartir Datos de Regulación

La mejora de mayor envergadura es permitir que desde una aplicación Taimun Mobile se puedan compartir los datos de una regulación a otra. Para afrontar este problema decidimos usar las tecnologías cloud ofrecidas por Google, que ya detallamos anteriormente.

Planteamos el sharing de regulaciones de la siguiente manera: en la vista de cada regulación en la lista de regulaciones añadiremos un botón que permita subir sus datos. Una vez que los datos estén subidos, aparecerá en pantalla un código numérico que se puede compartir con otras personas para que puedan descargarla.

Por otra parte, habrá una opción de añadir una nueva regulación introduciendo un código. Si se introduce un código válido, se descarga la regulación asociada a dicho código, junto a todos sus datos asociados.



(a) Creación de nueva regulación (b) Edición umbral de pulsaciones (c) Nuevo umbral definido

Figura 3.3: Compartir regulaciones. Las regulaciones pueden ser subidas a la Cloud con el icono de la figura 3.3(a). Al hacerlo, obtenemos el mensaje de la figura 3.3(b), en el cual se muestra el código para su posterior descarga. Con ello, cualquier usuario puede descargársela al introducir el código, como se muestra en la figura 3.3(c)

A las 24 horas de generar el código, los datos se borran y ya no estarán accesibles. Si se desea volver a compartir la regulación nuevamente, habrá que generar un nuevo código. Adicionalmente, una vez generado el código, si se realizan cambios en la regulación y se desea volver a compartir, será necesario generar otro código. Esta decisión de diseño la hemos tomado para simplificar el proceso de compartir, puesto que actualmente Taimun no implementa ningún sistema de gestión de identidad de usuarios (entendidos como los guardianes que usan la app para la monitorización de personas con TEA).

4

Desarrollo

4.1. Inspección de Código

Comenzaremos esta sección elaborando los pasos que seguimos para restaurar la funcionalidad de la app, de manera que ésta pueda ser publicada nuevamente en Play Store.

El primer paso fue la inspección general de código, para realizar un mapa mental general de la arquitectura interna de la App Mobile y Wear.

Por otra parte, en Taimun-Watch encontramos una arquitectura altamente asíncrona con gran parte del código ejecutado en hilos secundarios. Wear entra a primer plano activando MainActivity, que prepara el botón «START» y parsea el JSON recibido con la regulación emocional.

Una vez que el usuario presiona el botón «START» se inicia un hilo asíncrono que prepara el dispositivo para la medición del pulso cardíaco. Una vez que los sensores están preparados, se lanza otro hilo que recoge las medidas periódicamente y las compara con un umbral. En caso de que se requiera la intervención de una regulación emocional, se transmite una señal que es recibida por un BroadcastReceiver estático registrado en MainActivity que se encarga de finalizar MainActivity y de llamar a AssistActivity, que muestra la regulación en cuestión. Como cada regulación está compuesta por varias estrategias de distintos tipos, se llama a las clases apropiadas que muestran las estrategias pertinentes. Una vez que la regulación ya no tenga más estrategias el hilo principal vuelve a MainActivity, y el hilo medidor sigue registrando medidas periódicamente.

Opinamos que una comprensión profunda del código sería crítica para resolver los errores que se pudieran presentar, así como para añadir funcionalidad adicional a un código fuertemente acoplado, por lo que dedicamos una gran cantidad de tiempo y esfuerzo a obtener dicho conocimiento. Comenzamos con la inspección de código del Wear, al tener una arquitectura más sencilla.

4.1.1. Funcionalidad de Wear

MainActivity

MainActivity es el punto de entrada de la aplicación y su primera tarea, realizada en onCreate, es parsear el JSON recibido desde Mobile para obtener los datos necesarios que alimenten una regulación. Posteriormente se registra un onClickListener que define el comportamiento para el botón «START/STOP». Para el botón «START», que inicia una asistencia, se realiza la llamada a SensorService, un servicio que gestiona la activación de los sensores de medición.

SensorService

Este servicio, junto con SensorAdmin, gestiona la activación y desactivación de los sensores. Los sensores son considerados como una fuente de información a la que se acopla un observable. En el caso de Taimun, el observable es el propio SensorService, que implementa la interfaz Observer.

SensorService, en su método onDestroy, realiza llamadas a SensorAdmin para destruir los sensores registrados. Por lo tanto, las mediciones dejan de realizarse cuando SensorService se destruye. Constituye entonces el nexo que une el hilo de ejecución de la App con la gestión de los sensores y sus mediciones.

SensorAdmin

Los sensores en sí, como miembros de una clase, se encuentran en SensorAdmin. SensorService realiza llamadas a SensorAdmin para modificar el comportamiento de los sensores. Sensor Admin posee para cada sensor implementaciones de la clase abstracta ObservableSensor, y es aquí donde finalmente encontramos el acceso directo a los sensores. Adicionalmente, cada ObservableSensor posee una SensorMeasure, donde SensorAdmin registra las mediciones a medida que se realizan para su posterior procesamiento e inclusión en las estadísticas.

Registrado como Observable en los Sensores, SensorService llama a SensorAdmin.update() estableciendo una relación bidireccional entre ambas clases que originalmente resultó difícil de comprender. Esta función es la que registra las mediciones en ObservableSensor. Adicionalmente, run() crea y ejecuta un *thread* llamado HiloCalculadorClasificador.

HiloCalculadorClasificador

Este hilo es llamado cada vez que se registran nuevas mediciones por parte de los sensores, y su principal responsabilidad es obtener una media de las últimas mediciones para poder determinar si hay que comenzar el inicio de una regulación. La regulación se inicia con el broadcast de un mensaje.

StartAssistReceiver

Este Receiver, declarado como clase estática en MainActivity, responde a los mensajes enviados por HiloCalculadorClasificador e inicia una AsyncTask que prepara e inicia la regulación.

StartAssist

Esta AsyncTask es la que inicia una nueva regulación o continúa una regulación ya existente. Para eso accede al objeto creado a partir del parseo anterior del JSON ya recibido. Cada regulación está compuesta de una serie de estrategias, y si la regulación actual todavía tiene estrategias pendientes, StartAssist carga la siguiente estrategia y finaliza la MainActivity. Si ya no hay más estrategias, se vuelve a iniciar MainActivity. Una vez que el HiloCalculadorClasificador detecte que los criterios establecidos vuelven a indicar la necesidad de ejecutar la regulación, StartAssist volverá a ejecutarse.

Regulation

Esta clase es creada a partir de la información recibida de Mobile, en la que se incluyen las estrategias en sí e información adicional sobre cómo presentarlas, como el tipo de estrategia y el tiempo que cada estrategia va a ser mostrada en pantalla.

Estrategia

Clase que guarda los detalles concretos de la estrategia, incluidos los pictogramas que se van a mostrar, así como detalles sobre el tipo temporal y estático.

AssistActivity

Esta clase resuelve la necesidad de mostrar las estrategias en pantalla. Es una clase que es extendida en varias clases, como ConfirmationActivity, ContentPictureActivity, GifActivity, MultipleSelectorActivity, etc.

4.1.2. Funcionalidad de Mobile

La principal tarea de Mobile consiste en asistir la creación y sincronización de estrategias y regulación. Para ello, se emplean una serie de clases y adaptadores que describiremos a continuación. El punto de entrada de la App es MainActivity, que define el comportamiento de los botones del menú lateral. La primera pantalla que ve el usuario, viene implementada por HomeFragment, y permite seleccionar entre la lista de estrategias, implementada por StrategyFragment y StrategiesDialogFragment, y la lista de regulaciones, que responde a RegulationFragment. Dentro de las estrategias, una vez

seleccionada una, nos encontramos con una vista de detalle apoyada en la clase StrategyViewAdapter. Los distintos parámetros de la estrategia se definen realizando uso de StrategyFirstTabFragment y StrategySecondTabFragment. Internamente, una estrategia puede estar compuesta por varios pasos, y estos se definen en SubmenuFirstTabFragment, SubmenuSecondTabFragment y SubmenuThirdTabFragment. Nuevas estrategias y regulaciones son gestionadas por NewRegulationActivity y NewStrategyActivity.

El acceso a la base de datos SQL se realiza mediante la clase DatabaseAdapter y la comunicación con la Data Layer mediante la clase GoogleConnection, mientras que el acceso a la cámara está gestionado por PhotoUtils.

Las estadísticas, que son generadas a partir de los logs, utilizan las clases StatisticsFragment, StatisticsViewAdapter y StatisticsFragmentAdapter.

4.1.3. Estadísticas

Mobile permite visualizar de manera gráfica los datos biométricos de los usuarios de los relojes dentro del marco de una regulación. Para ello, recibe de Wear dichos datos en formato de log, como se expone en la siguiente sección. Una vez que los datos han sido recibidos, la clase StatisticsAdapter, funcionando junto al StatisticsFragment, parsea los ficheros de log creando las clases de Statistic necesarias y filtrándolas por fechas para mostrarlas. Las estadísticas se agrupan primero por días y una vez seleccionado el día por las distintas regulaciones empleadas.

4.1.4. Comunicación

Mobile y Wear se comunican realizando uso de la Data Layer API, que permite un flujo de datos bidireccional. El patrón de diseño es el mismo tanto en Mobile como Wear, distinguiéndose por la implementación concreta, por lo que comenzaremos hablando de las generalidades de ambas implementaciones y concluiremos con las diferencias entre ambas.

Recepción de Mensajes

Para poder realizar escuchas a paquetes de datos entrantes, se hace uso de un servicio que extiende a la clase WearableListenerService. Adicionalmente, implementa la interfaz GoogleApiClient-ConnectionCallbacks. Mediante la extensión de clase se tiene acceso al método onDataChanged() que se ejecuta cuando un paquete se recibe por primera vez, o cuando los contenidos del paquete han cambiado.

Cuando se obtiene un paquete se realizan una serie de comprobaciones antes de procesarlo. La primera se asegura que el paquete realmente vaya destinado al dispositivo. Esta comprobación se

realiza aprovechando el hecho de que tanto Mobile como Wear, cuando envían un paquete, añaden a la Uniform Resource Identifier (URI) definida el identificador del dispositivo destino. Conociendo esto, se parsea la URI para obtener el identificador.

Una vez que se ha comprobado que el paquete va dirigido al dispositivo, se comprueba que la URI esté dentro de las reconocidas. Si no lo está, no se procede con el procesamiento. Si la URI sí es reconocida, los datos se extraen y se transmiten a la parte responsable de su tratamiento.

Envío de Mensajes

Los mensajes no se pueden enviar desde el hilo de ejecución principal, puesto que esto causaría una ralentización del rendering gráfico. Para solventar el problema, se hace uso de un hilo secundario encargado de realizar la emisión: SendToDataLayerThread. Este hilo recibe como argumento el DataMap que se quiera enviar y la URI, y llama a Wearable.DataApi, que requiere un .await() por ser asíncrona.

Transmisión de Regulaciones

El principal uso de la comunicación es transmitir una regulación emocional desde Mobile hasta Wear. Como una regulación puede estar compuesta por varias estrategias, y cada una tiene una imagen o texto asociado, no basta con realizar la transmisión de un único paquete, sino que se realiza en varias emisiones.

Por parte de Wear, la sincronización comienza cuando recibe un mensaje de tipo MOB_TO_WEAR. Al recibirlo, se borra la regulación previa y se envía el mensaje READY_TO_SYNC_REGULATION, que indica que el Wear está preparado para recibir los datos de la regulación. Mobile envía el primer archivo con el mensaje FILE junto con los primeros datos. Wear, si parsea correctamente los datos enviados, responde con SEND_NEXT_IMAGE_FILE, o en caso de no procesar correctamente los datos, envía SEND_IMAGE_FILE_AGAIN. Cuando Mobile ya ha enviado todos los archivos, envía un último mensaje NO_MORE_FILES, que señaliza el fin de la comunicación.

La secuencia de mensajes, si no ocurre ningún error, es entonces:

- Mobile: MOB_TO_WEAR
- Wear: READY_TO_SYNC_REGULATION
- Mobile: FILE
- Wear: SEND_NEXT_IMAGE_FILE
- [...Repetición de envío...]
- Mobile: NO_MORE_FILES

Envío de Logs

El otro uso que tiene la Data Layer API es el envío de datos biométricos desde Wear a Mobile, para poder mostrar estadísticas de las regulaciones llevadas a cabo, junto a los datos biométricos registrados por el sensor.

El envío de los logs es comenzado por Mobile mediante el envío de START_SYNC_LOG_FILES. Wear responde obteniendo los datos de los logs y comprobando si se pueden enviar de un sólo mensaje. Si este es el caso, responde con un paquete LOG_FILE que lleva como atributo especial IS_FINAL_PART_KEY = true.

Si el tamaño de los logs es mayor del que se puede enviar en un único paquete, el envío del paquete lleva IS_FINAL_PART_KEY = false. Cuando mobile se encuentra en esta situación, responde con el mensaje SEND_LOG_FILE. Esta secuencia se repetirá hasta que se envíe el último paquete de logs. Al igual que en el caso de envío de regulaciones, Mobile puede responder con el mensaje SEND_LOG_FILE_AGAIN en caso de que no haya podido procesar correctamente el paquete.

4.2. Adaptación del Código

4.2.1. Migración a AndroidX

Una vez concluida la inspección de código, intentamos realizar un primer despliegue de la aplicación móvil. Lamentablemente, la aplicación *crasheó* tras mostrar los créditos iniciales.

Esto reveló el primero y más inmediato de los errores que debíamos de resolver: el uso de librerías anticuadas y no compatibles con la API de Android presente en la mayoría de los dispositivos móviles actuales.

Comenzamos entonces nuestra adaptación de la app encontrando instancias en las que en los layouts se usaron elementos eXtensible Markup Language (XML) anticuados. La mayoría de los layouts presentaba al menos un elemento de dicho tipo, lo cual mandó una inspección fichero a fichero.

En la mayoría de los casos, los nuevos elementos se parecían a los anticuados, e incluso en algunos simplemente había que cambiar la ruta o el paquete del cual se obtenían. Esto estaba causado, como nos dimos cuenta a medida que realizábamos las actualizaciones, porque la librería AppCompat había sido sustituida por AndroidX. Aun así, se presentaron casos en los que hubo que buscar cuál era la correspondencia exacta entre elementos.

Al igual que los elementos XML, las rutas de paquetes que provenían de AppCompat en los archivos .java también tuvieron que ser cambiadas. Aproximadamente en la mitad de los casos, bastó con cambiar la ruta de importación, pero otros casos requerían soluciones individuales que en ocasiones

sólo respondían al método «guess and check». Por ejemplo, un error se solucionó eliminando ciertos «nullable parameters» de la función que se estaba extendiendo. Aunque los cambios en sí parecen sencillos, la escala del proyecto mandaba hacerlos con cautela para evitar errores.

Para poder usar las nuevas librerías hubo que cambiar también las «coordenadas» de Gradle para que importen las nuevas librerías.

4.2.2. BroadcastReceiver Globales

Como se ha indicado previamente, la regulación emocional se inicia cuando un BroadcastReceiver obtiene un mensaje desde el HiloCalculadorClasificador. Una vez que logramos que la app Mobile funcionara y que sincronizara datos con el Wear, nos encontramos con el problema de que a pesar de que el umbral se superase, no se iniciaba la regulación. Para resolver el problema, marcamos con el debugger los puntos que sabíamos gracias a la inspección de código que deberían de alcanzarse para disparar una regulación, y observamos el comportamiento.

Pudimos comprobar que a pesar de que efectivamente el HiloCalculadorClasificador emitiera el mensaje, el BroadcastListener, BL, no se ejecutaba. Después de asegurarnos que no hubiera ningún problema con el filtro de mensajes del BL, consultamos la documentación de Android, donde se indicaba que los BroadcastListeners globales, que son aquellos declarados en el manifest, sólo pueden recibir intents explícitos. El mensaje de inicio de regulación era un intent implícito, por lo que no podíamos usar un BL global para responder.

Para solucionar el problema, cambiamos la declaración de BroadcastListener, manejando su ciclo de vida de manera apropiada para asegurarnos de que pueda responder cuando las mediciones se hayan comenzado a realizar.

4.2.3. Estadísticas

El último gran desafío a la hora de restaurar la funcionalidad ya existente fueron las estadísticas. Una vez que habíamos comprendido su funcionamiento gracias a la inspección del código, pudimos verificar que la causa por la que no funcionaban no se debía a un error a la hora de mostrar los datos, si no a la solicitud y recepción de los logs.

Realizamos varias pruebas asistidos del debugger, y llegamos a la conclusión de que las estadísticas se sincronizaban de manera errática principalmente porque Mobile no solicitaba los logs cuando debería. Nuestro primer intento de arreglo fue introducir solicitudes de sincronización automáticas en puntos de ejecución donde tuvieran sentido, por ejemplo al finalizar una regulación, pero después de comprobar que la sincronización automática de logs dejaba mucho que desear, decidimos trasladar la funcionalidad a un botón.

El botón, incluido en la pantalla principal, realiza la tarea de solicitar los logs mediante la Data Layer API. La recepción de los logs es gestionada, como se explicó anteriormente, por el servicio de escucha de Mobile.

4.2.4. Imágenes

Para que las imágenes funcionasen, debido al aumento del SDK mínimo, debimos de cambiar sus URIs. Además, con las últimas versiones de Android, una aplicación no puede acceder a la cámara o al almacenamiento interno de un dispositivo sin que el usuario lo haya autorizado de manera explícita. Por ello, para su adaptación, tuvimos que realizar las siguientes modificaciones:

- Cambio de provider en Manifest, y creación de fichero con el path, como puede verse en la figura 4.1
- Ajustes en los permisos, presentes en la figura 4.2

```
<provider
    android:name="androidx.core.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths"/>
</provider>
```

Figura 4.1: Fragmento Android Manifest

```
try {
    if (opcion.equals("Cámara")) {
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA) != PackageManager.PERMISSION_GRANTED) {
            // No explanation needed; request the permission
            ActivityCompat.requestPermissions( activity: this,
                    new String[]{Manifest.permission.CAMERA},
                    MY_PERMISSIONS_REQUEST_CAMERA);
        }
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.WRITE_EXTERNAL_STORAGE) != PackageManager.PERMISSION_GRANTED) {
            // No explanation needed; request the permission
            ActivityCompat.requestPermissions( activity: this,
                    new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                    MY_PERMISSIONS_REQUEST_WRITE_EXTERNAL_STORAGE);
        }
        if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED &&
                ContextCompat.checkSelfPermission( context: this, Manifest.permission.WRITE_EXTERNAL_STORAGE) == PackageManager.PERMISSION_GRANTED) {...}
```

Figura 4.2: Permisos almacenamiento y cámara

4.3. Umbral Personalizado

Uno de nuestros objetivos de nuevo desarrollo era añadir la posibilidad de personalizar el umbral de pulsaciones para la regulación emocional. De esta manera, distintas regulaciones tendrán distintos umbrales, según dicte su naturaleza particular.

El desarrollo de esta funcionalidad se divide en su parte Mobile, en la que se indica el umbral deseado, y el desarrollo Wear, donde el umbral surte su efecto.

4.3.1. Desarrollo Mobile

La primera parte del desarrollo en mobile fue ampliar la estructura de datos con el dato del umbral. Internamente, las regulaciones se guardan en una base de datos SQL, por lo que ampliamos la definición de la tabla Regulación para que almacenase el nuevo parámetro.

Posteriormente fue necesario modificar la definición de la clase Regulación, incluyendo su creador y los getter/setter correspondientes. Al cambiar la definición del creador, tuvieron que modificarse distintas ocurrencias en las que se le hace referencia.

El umbral es introducido en la Card de creación/modificación de regulación. Añadimos a la lógica del onClickListener el umbral introducido, que posteriormente lo guarda en la base de datos SQL. Una vez que un umbral esté introducido, se obtiene y muestra en la tarjeta para su visualización y potencial modificación. Cabe destacar que la primera vez que se accede a la tarjeta de una regulación, cuando se está creando, el umbral tiene un valor por defecto de 120 pulsaciones por minuto.

Finalmente, el umbral se añade a la estructura JSON que se transmite por bluetooth al Wear.

4.3.2. Desarrollo Wear

Una vez que se recibe el JSON, añadimos el parseo adicional para obtener el umbral. Para garantizar máxima extensibilidad guardamos el umbral como una sharedPreference de manera que pueda ser leído por hilos asíncronos sin pasarlo como argumento. En nuestro caso, se modifica la lógica condicional en HiloCalculadorClasificador para iniciar la actividad sólo cuando las pulsaciones sean superiores al umbral.

4.4. Control Remoto

El objetivo de este desarrollo es permitir que desde el móvil se inicien y finalicen asistencias, entendidas como una medición cíclica que en casos de alteración emocional conlleven la ejecución de la

regulación seleccionada.

4.4.1. Desarrollo Mobile

Comenzamos añadiendo dos botones principales al layout principal. Estos botones servirán para iniciar y finalizar la asistencia de manera remota. Los clicklisteners asociados realizan una llamada a los componentes mobile que se encargan de la comunicación con el wear.

Para comunicarse con el wear debemos de establecer una serie de parámetros. El principal es una URI que identifica el mensaje mandado. También se añade un DataMap que contiene los pares clave:valor que queramos pasar. Para nuestro problema particular decidimos crear dos URIs distintos /start y /stop. Al realizar la llamada a la clase interna, se añade a la URI el identificador individual del Wear al cual el mensaje va dirigido. Como la información crítica va codificada en la propia URI introducimos en el DataMap un timestamp comunicar el momento en el que se ha realizado la llamada.

4.4.2. Desarrollo Wear

El desarrollo comienza con la recepción del mensaje, en cuyo código debemos de añadir dos nuevos casos al switch que se encarga de gestionar los mensajes recibidos. Esto ocurre en un servicio llamado ListenerService. Optamos por un desarrollo asíncrono donde la recepción del paquete dispara el envío de un mensaje que posteriormente es manejado por otras partes de la app para realizar los efectos apropiados.

Si se recibe un mensaje de inicio de asistencia cuando ya hay una iniciada, no se emite ningún mensaje de inicio. El mismo comportamiento ocurre con los mensajes de parada si no hay ninguna regulación activa. Esto dota de idempotencia a los botones de comienzo y parada en el móvil.

Los mensajes son recibidos por dos BroadcastListeners situados en MainActivity y se inician los pasos necesarios para comenzar o parar la asistencia. En caso de iniciar se siguen los siguientes pasos:

- Bloqueo de la pantalla
- Cambio del estado del botón de «START» a «STOP»
- Inicio de la asistencia mediante el broadcast de otro mensaje

Mientras que para la finalización de la asistencia se siguen los siguientes pasos:

- Parada del Hilo Calculador Clasificador
- Parada del sensor de pulso cardíaco
- Cambio del estado del botón de «STOP» a «START»

4.5. Compartir Regulaciones

Para poder compartir las regulaciones comenzamos desarrollando la capacidad de subir una regulación, junto con todos sus datos asociados a Google Cloud. Continuamos con el desarrollo de un sistema que asocie a cada regulación subida un código único. Posteriormente debemos de implementar la funcionalidad de obtener los datos de la regulación de Google Cloud, y guardarlos en la base de datos SQL de la aplicación. Adicionalmente debemos de implementar un microservicio que elimine una regulación una vez pasadas las 24 horas para que no se acumulen las regulaciones en la nube.

4.5.1. Subida de Regulaciones

El primer paso para poder subir las regulaciones es disponer de la información en un formato fácilmente almacenable. Para ello empleamos un formato JSON que defina una regulación. Las imágenes se identifican por el nombre con el que estén guardadas en el sistema de ficheros locales de la aplicación.

Comenzamos con la subida del fichero JSON a Firebase. Generamos para cada regulación que hemos subido un identificador que será la clave con la que la identificaremos.

Las imágenes se suben a Google Cloud Storage con la siguiente ruta /<código>/<nombre de imagen>. De esta manera cada imagen dentro de una regulación así como en todo el almacenamiento cloud queda identificado de manera única.

Colocamos en la view de la lista de regulaciones un botón a cuyo onClickListener asociamos la funcionalidad de subida. El primer paso es acceder a la información de la base de datos y construir el JSON de manera coherente a partir de los datos.

Una vez que el JSON y las imágenes han sudo subidas, se muestra una alerta al usuario indicándole en caso de éxito el código que debe compartir. En caso de que en algún paso de la subida ocurra un error, se informará al usuario mediante una alerta. El flujo interno a la hora de subir una regulación es:

- Seleccionar una regulación, que activa el onClickListener
- Preparación el JSON a partir de los datos de la regulación
- Se localizan las imágenes necesarias
- Se obtiene un código que será el identificador de la regulación
- Subida del JSON a Firebase
- Subida de las imágenes a Google Cloud Storage
- Se muestra el código al usuario

4.5.2. Descarga de Regulaciones

Para la descarga de regulaciones comenzamos añadiendo un input de texto en el cual se pueda introducir el código de regulación al pulsar sobre el botón de descarga. Una vez introducido el código, se descarga de Firebase el JSON, y se descargan las imágenes referenciadas, que son guardadas en el sistema de ficheros de la aplicación.

Cuando ya se hayan descargado todos los datos, se procede a convertir el JSON a una serie de consultas SQL para introducir primero la regulación en sí, y luego todas sus estrategias asociadas en la base de datos. Así, las acciones a realizar para descargar una regulación son:

- El usuario introduce el código
- Descarga del JSON utilizando el código
- Descarga de las imágenes combinando el código con los datos del JSON
- Almacenamiento de las imágenes
- Parseo del JSON en regulación y estrategias asociadas
- Guardado en la base de datos SQL la regulación y las estrategias

4.5.3. Borrado de Regulaciones

Para evitar que la base de datos se sobrecargue con información de regulaciones antiguas, desarrollamos un microservicio basado en NodeJS al cual vinculamos un trigger cada 24 horas para que recorra todas las regulaciones subidas y borre las que tienen más de un día de antigüedad. El microservicio se implementa en una Cloud Function la cual tiene acceso exclusivo de borrado de datos.

En trigger es otro servicio ofrecido por Google Cloud que admite una sintaxis CRON y que permite realizar llamadas HTTPS arbitrarias cuando tenga que ejecutarse.

En cuanto a la función, usamos la librería ofrecida por Google que permite comunicación directa con los otros componentes internos de Google Cloud, entre los que se incluye Firebase.

4.5.4. Configuración de Firebase

Establecemos finalmente las reglas de seguridad de Firebase. Permitimos que la creación y lectura se puedan realizar sin autenticación, pero prohibimos la edición y el borrado. La Cloud Function está exenta de estas limitaciones al usar la librería de Google junto con nuestros credenciales *full access*.

5

Integración, pruebas y resultado

En esta sección introducimos las pruebas que realizamos para verificar que el comportamiento de nuestra app se ajusta al esperado y planeado en las fases iniciales. En concreto, comenzamos probando que mediante la adaptación a las versiones modernas de Android se hayan preservado las funciones ya existentes de Taimun. Posteriormente verificamos que las mejoras añadidas desempeñen sus tareas correctamente.

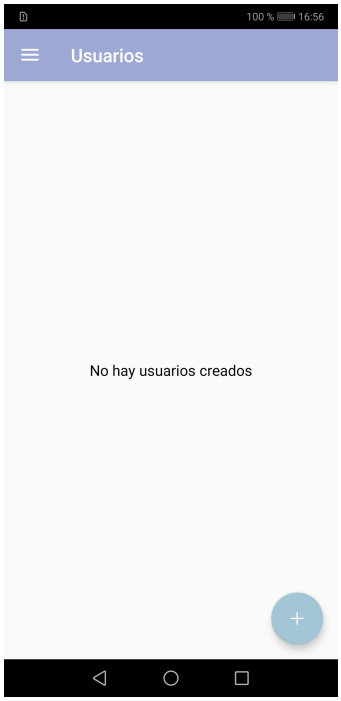
5.1. Actualización y adaptación

Creación de usuario y vinculación con Wear

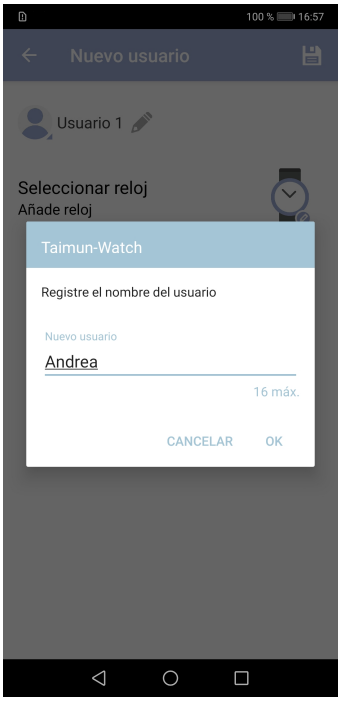
Nuestra primera prueba consiste en la creación de un usuario desde Mobile, la vinculación de dicho usuario con nuestro Wear Vapor 2 y la verificación de que la conexión se ha realizado correctamente mediante el envío de una regulación. Para asegurarnos de que la regulación se haya recibido, pulsamos el botón START en el watch. Si el paquete JSON y los datos asociados no se han recibido, la app Wear nos alertará. De manera concisa, los pasos de la prueba son:

- Creación de usuario
- Vinculación con wear
- Sincronización regulación
- Inicio de regulación en wear
- Comprobación estadísticas

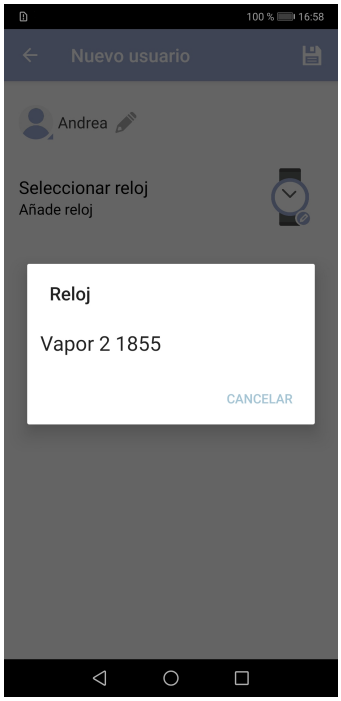
Las pruebas se desarrollaron satisfactoriamente, como se puede comprobar en la figura 5.1 adjunta.



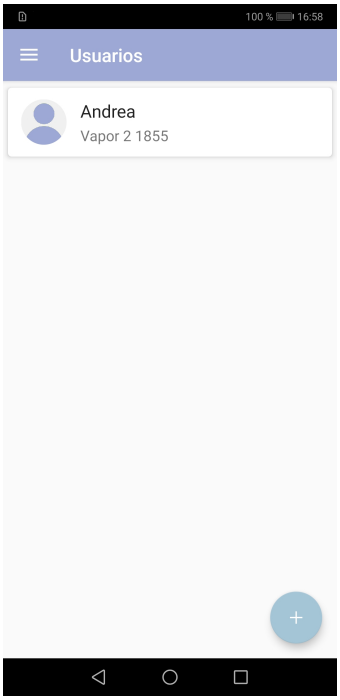
(a) Pantalla Usuarios



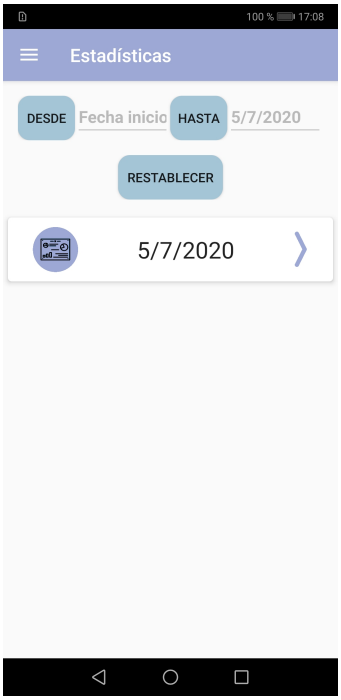
(b) Creación de usuario (nombre)



(c) Adición de reloj



(d) Usuario creado



(e) Estadísticas

(f) Estrategia regulación

Figura 5.1: Creación de usuario, vinculación con Wear e inicio regulación. Al hacer click en el botón «+» de la figura 5.1(a) se comienza la creación de un usuario, para el cual se selecciona nombre y reloj en las figuras 5.1(b) y 5.1(c) respectivamente. Al hacer click sobre el icono de guardado, se registra correctamente. Para completar la prueba, enviamos una regulación, y se registra correctamente como se puede ver en 5.1(e) y 5.1(f)

Correcto funcionamiento botón START/STOP

En esta prueba queremos comprobar que el botón «START» inicie los sensores y que al pulsar «STOP» las mediciones se dejen de realizar. Para ello observamos que el medidor de pulsaciones cardíacas funciona emitiendo un haz de luz y midiendo la dispersión de la misma que es conocida y predecible para la sangre humana. Por lo tanto, si el sensor está encendido, será visible una luz en la parte posterior del reloj. Los pasos para realizar esta prueba son:

- Pulsar el botón START
- Comprobar que el sensor emite luz
- Pulsar el botón STOP
- Comprobar que se deja de emitir luz desde el sensor

Comprobamos que tras pulsar START y esperar aproximadamente 3 segundos se comenzaba a emitir luz verde desde el sensor. El inicio de la emisión de luz coincide con el mensaje por terminal que añadimos para indicar que los sensores comenzaban a medir. Al pulsar STOP la luz deja de emitirse.

(a) Start (b) Sensor funcionando (c) Stop (d) Sensor apagado

Figura 5.2: Funcionamiento del botón START/STOP. Al hacer click en el botón de START visible en la figura 5.2(a) el sensor se pone en funcionamiento emitiendo luz, como se ve en 5.2(b). Al hacer click sobre el botón de STOP de 5.2(c), el sensor se apaga como puede verse en la figura 5.2(d)

Apropiado inicio y final de regulación

Continuamos nuestras pruebas comprobando que las regulaciones se inicien sólo cuando las circunstancias lo manden, y que no se inicien cuando no sea necesario. Es decir, estamos comprobando que una regulación se inicie si y sólo si se supera el umbral. Para ello hemos sincronizado una regulación muy sencilla que consiste de una única estrategia estática. El comportamiento esperado es que la regulación se inicie sólo cuando supere el umbral y no se inicie si el umbral no ha sido superado. Para medir el umbral cardíaco de manera independiente hemos usado un oxímetro de pulso, que también realiza mediciones de pulso cardíaco. Los objetivos de esta prueba son:

- Comenzar con un pulso sub-umbral y verificar que no se inicia regulación

- Inducir un pulso super-umbral mediante esfuerzo físico y verificar que la regulación se inicia
- Reducir el pulso mediante respiración diafragmática y comprobar que la regulación no se inicia, aunque el reloj esté midiendo durante un largo periodo de tiempo (10 minutos)
- Finalmente, volver a inducir un pulso alto y comprobar que se vuelve a mostrar la misma regulación

Gracias a la medición independiente con el oxímetro, pudimos comprobar que los resultados obtenidos coinciden con los esperados.

Comprobación de distintos tipos de estrategias

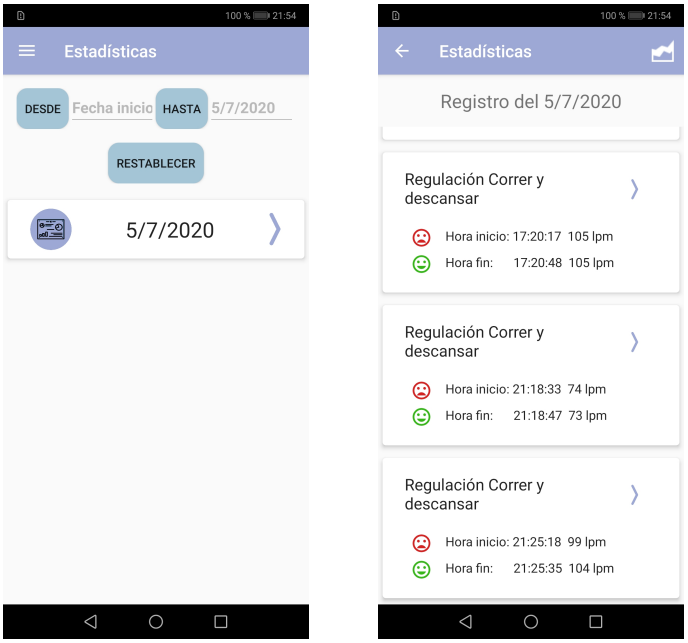
Pretendemos probar también que los distintos tipos de estrategias que ofrece Taimun se presentan de manera correcta, para ello creamos distintos tipos de estrategias, cada una representativa de un tipo de AssistActivity distinto, y las cargamos secuencialmente al reloj para poder comprobarlas individualmente. El desglose de las clases comprobadas para la prueba es el siguiente:

- Time Activities
  - TimePicture
  - CarruselActivity
  - TimeGif
- Static Activities
  - GifActivity
  - MultipleSelectorActivity
  - ConfirmationActivity
  - ContentPictureActivity
  - PositiveReinformentActivity

Con esta prueba, podemos concluir que todas las estrategias, tanto las temporales como las estáticas, funcionan de la manera esperada.

Estadísticas

Como se explicó en la sección de Desarrollo, las estadísticas no se sincronizaban de manera adecuada. Para solventarlo, hay un botón en la pantalla Home con el cual las mediciones del usuario wear activo se sincronizan de manera correcta con el dispositivo móvil. Para la prueba, se realizaron varias mediciones durante un día utilizando la misma estrategia, para ver si los datos se almacenaban y representaban de manera correcta. Los resultados se pueden ver en la figura 5.3.



(a) Historial de estadísticas (b) Estadísticas del 5/7/20

Figura 5.3: Estadísticas. En la figura 5.3(a), se pueden ver el registro de estadísticas. Al hacer click sobre el día 5/7/20 se pasa a la figura 5.3(b), en la cual se pueden ver las pulsaciones registradas al inicio y fin de las regulaciones

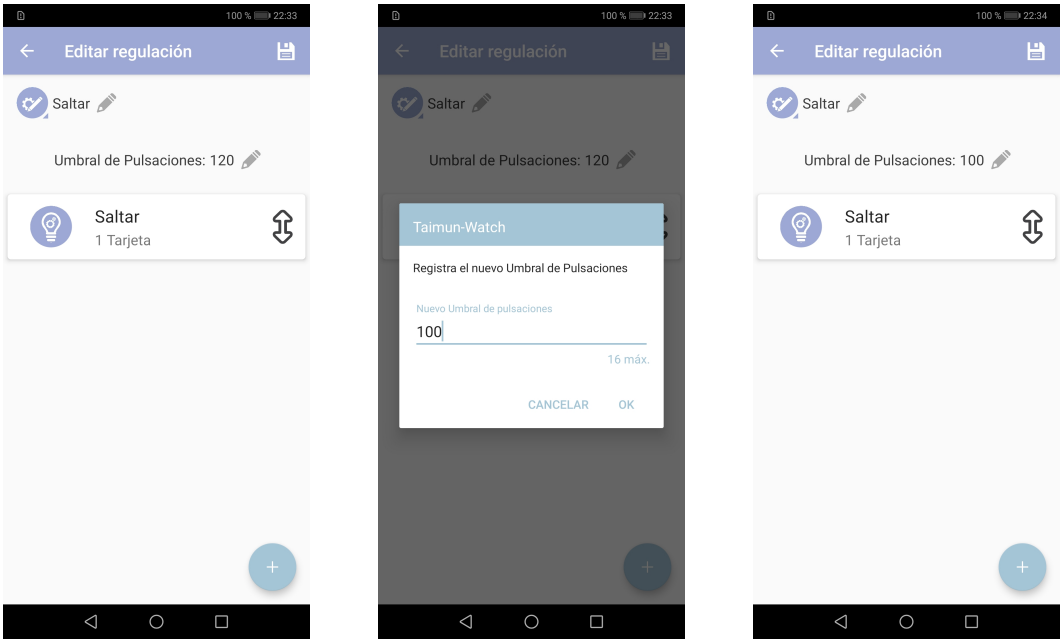
5.2. Umbral personalizable

En esta sección nuestro objetivo es comprobar que la funcionalidad adicional de umbral personalizable que añadimos se comporta de manera esperada. Para ello definimos distintas regulaciones con distintos umbrales y mediante nuestro oxímetro de pulso comprobamos si los umbrales disparan la regulación cuando deberían. Para incrementar el pulso de manera gradual se utilizó una cinta de correr a la cual aumentamos la intensidad de manera escalonada. Las pruebas consistieron entonces en:

- Carga de Regulación con Umbral <X>
- Aumento de las pulsaciones del sujeto hasta <X> + 5
- Comprobar que la regulación se inicia
- Cargar otra regulación con Umbral <X> + 10 y volver a empezar

En total definimos 7 regulaciones con los umbrales variando desde 80 a 140 pulsaciones y en todos los casos el comportamiento fue el esperado.

En la figura 5.4 se ve cómo las pulsaciones se actualizan de manera correcta.



(a) Regulación con umbral 120 (b) Cambio del umbral a 100 (c) Regulación con umbral 100

Figura 5.4: Umbral personalizable. Comenzamos con una regulación, Saltar, cuyo umbral de pulsaciones es 120 5.4(a), y con el botón de edición lo cambiamos a 100 5.4(b). Al confirmar el cambio, el umbral se actualiza correctamente, como puede comprobarse en la figura 5.4(c)

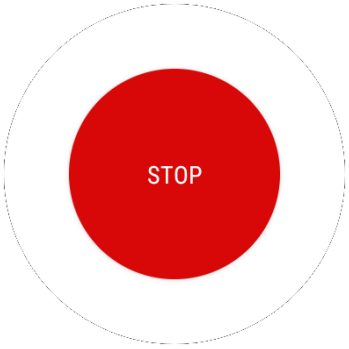
5.3. Inicio y fin de asistencia remoto

Finalmente, en estas pruebas nuestro objetivo fue comprobar que la asistencia se puede iniciar y parar de manera remota. Para ello cargamos una regulación en el Vapor 2 y lo bloqueamos. Posteriormente iniciamos y terminamos varias veces la asistencia. Los pasos concretos que seguimos fueron:

- Carga de estrategia y bloqueo de reloj
- Comienzo remoto y comprobación de que el reloj empieza a medir
- Volver a pulsar el botón de inicio remoto y verificación de que el reloj continúa su asistencia sin interrupción ni volver a empezar
- Pulsar el botón de fin remoto y observar que la asistencia se para
- Volver a pulsar el botón fin y comprobar que no hay cambios en el reloj.

Todas las comprobaciones funcionan de manera correcta, y los botones de inicio y fin exhiben el comportamiento idempotente esperado. Adjuntamos un ejemplo de funcionamiento en la figura 5.5.

(a) Asistencia iniciada



(b) Medición en marcha

(c) Asistencia finalizada

(d) Medición detenida

Figura 5.5: Inicio y fin de asistencia remoto. El usuario de la aplicación Mobile hace click sobre el icono para iniciar asistencia como se muestra en 5.5(a), con lo que la medición se inicia en el reloj 5.5(b). Cuando desea finalizar, hace click sobre el botón de la figura 5.5(c), con lo que el reloj detiene las mediciones 5.5(d)

5.4. Compartición de regulaciones

Para verificar que los datos de una regulación se puedan compartir entre dos dispositivos móviles con Taimun instalados realizamos varias pruebas. Comenzamos con Unit Tests en los que verificamos que las funcionalidades de subir datos a Firebase y de descargar dichos datos funcionen correctamente. Posteriormente realizamos pruebas de integración, comprobando que el código generado al subir a Firebase una regulación posteriormente permite descargarse la misma regulación. Adicionalmente, comprobamos que en la ausencia de conectividad a internet la app no dejara de funcionar. Análogamente, comprobamos que si se introduce un código que no se corresponde a ninguna regulación, la ejecución de la app continúa normalmente.

La primera prueba consistió en subir varias regulaciones a Firebase. La primera regulación que subimos constaba únicamente de una estrategia. Después de comprobar que se había depositado correctamente en la base de datos, procedimos a subir una regulación con varias estrategias, para comprobar que todas se subieran correctamente.

Posteriormente, descargamos una regulación de una sola estrategia en otro dispositivo móvil, y comprobamos que se podía ejecutar correctamente. Repetimos el procedimiento con una regulación de varias estrategias y comprobamos que la regulación ejecutaba correctamente todas las estrategias, mostrando las imágenes asociadas.

Finalmente, realizamos pruebas de integración en las que, después de haber instalado Taimun en dos móviles distintos, procedimos a subir a la nube dos regulaciones, y luego a descargar la regulación en el otro dispositivo. Este test «crossover» nos garantiza que la integración de las funcionalidades necesarias para compartir funcionan correctamente.

La prueba presente en la figura 5.7, corresponde a la compartición de una regulación entre dos dispositivos móviles. También se adjunta la figura 5.6, en la cual se ve la regulación de dicha prueba subida a Firebase.

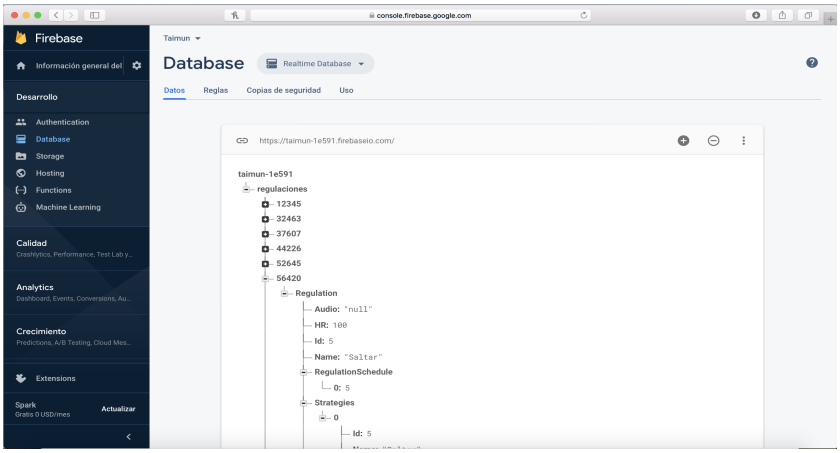
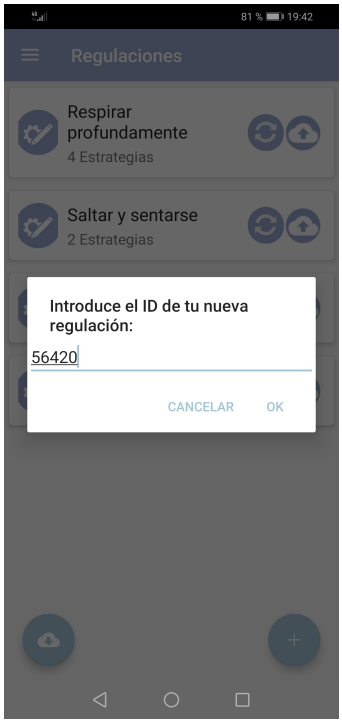


Figura 5.6: Base de datos Firebase

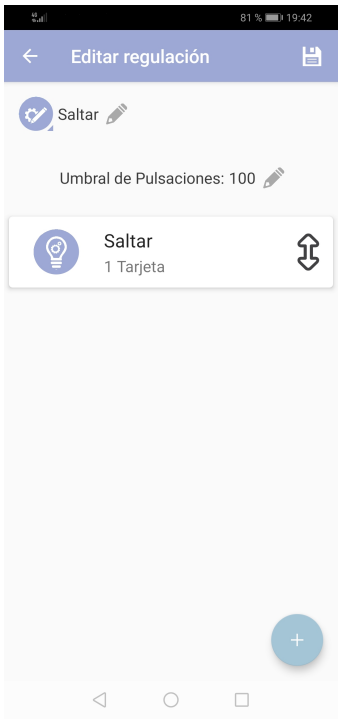
(a) Teléfono A: Regulaciones

(b) Teléfono A: Código regulación



(c) Teléfono B: Código regulación

(d) Teléfono B: Regulaciones



(e) Teléfono B: Regulación compartida

Figura 5.7: Compartición de regulaciones. Con el primer dispositivo, teléfono A, vemos las regulaciones en la figura 5.7(a) y compartimos «Saltar», con lo que obtenemos el código presente en 5.7(b). Con el segundo dispositivo, teléfono B, la descargamos como se muestra en la imagen 5.7(c). Al hacer click sobre ella en 5.7(d), vemos que la regulación se ha descargado correctamente 5.7(e)

6

Conclusiones y trabajo futuro

6.1. Conclusiones

En este trabajo hemos actualizado una aplicación de asistencia emocional para personas con TEA. Comenzamos el problema mediante la comprensión de la estructura de la aplicación y del código, tanto de la parte Wear como de la Mobile. Una vez obtuvimos suficiente conocimiento, proseguimos con la identificación de los errores que impedían la ejecución. Los identificamos como incompatibilidades de librerías y usos de patrones de diseño ya no aceptados, como BroadcastListener globales.

Corregidos los errores, proseguimos con la ampliación de la funcionalidad de Taimun, añadiendo la capacidad de introducir un umbral personalizado, la posibilidad de comenzar y parar las asistencias emocionales de manera remota así como el sharing de regulaciones aprovechando Google Cloud. Esto aumenta la personalización y flexibilidad de la app, y es nuestra esperanza que contribuyan a la utilidad para las personas con TEA. Finalmente, subimos la aplicación a Play Store.

6.2. Trabajo futuro

Como posibles ampliaciones de la aplicación, nos gustaría sugerir la posibilidad de que el wear alerte al Mobile de que se ha superado el umbral de regulación emocional. De esta manera, un guardián de varias personas con TEA tendría una mayor visión global.

Por otra parte, sería interesante crear un nuevo tipo de asistencia que muestre sonidos, como por ejemplo un mensaje grabado, o producido con tecnología *Text To Speech*. De esta manera se ampliaría el abanico de posibilidades para la regulación emocional.

Planteamos también la posibilidad de implementar en Taimun un sistema de gestión de identidad, que en caso de ser implementado con el Identity provider de Google aumentaría la integración con Firebase y permitiría un sharing directo de regulaciones emocionales. Este sistema también aumentaría la seguridad y permitiría que se pudiera usar una misma cuenta en varios dispositivos, aprovechando la Realtime database de Firebase para mantener todos los datos sincronizados.

La posibilidad de que la tecnología se emplee como vehículo terapéutico con un impacto real refleja la idea de que el progreso del conocimiento no es un concepto abstracto, sino que tiene repercusiones reales sobre la calidad de vida de las personas que más lo necesitan. Estamos muy orgullosos de haber podrido contribuir a que Taimun vuelva a Play Store, y esperamos que tenga un impacto positivo sobre las personas con TEA.

Bibliografía

[1] "Android developer." https://developer.android.com.

[2] "Firebase." https://firebase.google.com.

[3] "Stack overflow." https://stackoverflow.com.

DEFINICIONES

estrategia Conjunto de tareas a realizar, que pueden ser representadas mediante imágenes o texto.

regulación Secuenciación de estrategias de control emocional.

wearable Dispositivo electrónico que se incorpora en alguna parte del cuerpo e interactúa de forma contínua con el usuario y con otros dispositivos con la finalidad de realizar alguna función concreta, como por ejemplo, controlar estado de salud del portador.

Acrónimos

SNC Sistema Nervioso Central.

TEA Trastorno del Espectro Autista.

URI Uniform Resource Identifier.

XML eXtensible Markup Language.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID